

A

NEW UNITED STATES UTILITY PATENT APPLICATION  
under 37 C.F.R. 1.53(b)

Page 1

Atty. Docket No. 03797.84196

CERTIFICATE OF MAILING UNDER 37 C.F.R. § 1.8: The undersigned hereby certifies that this Patent Application and all papers noted herein as being attached, are being deposited with the United States Postal Service today, **April 28, 2000**, as Hand Delivered Mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

Assistant Commissioner for Patents  
Box Patent Applications  
Washington, D.C. 20231

Enclosed herewith is a new patent application and the following papers:

First Named Inventor (or application identifier): Georgios Chrysanthakopoulos

Title of Invention: CREATION AND USE OF VIRTUAL DEVICE  
DRIVERS ON A SERIAL BUS

☒ Specification 17 pages (including specification, claims, abstract) / 36 claims (7 independent)

☒ Declaration/Power of Attorney is:

☒ attached in the regular manner.

☐ NOT included, but deferred under 37 C.F.R. § 1.53(f).

☒ 5 Distinct sheets of ☐ Formal ☒ Informal Drawings

☐ Preliminary Amendment.

☐ Information Disclosure Statement

☐ Form 1449

☐ A copy of each cited prior art reference

6. ☒ Assignment with Cover Sheet.

7. ☐ Priority is hereby claimed under 35 U.S.C. § 119 based upon the following application(s):

Country	Application Number	Date of Filing (day, month, year)

8. ☐ Priority document(s).

9. ☐ Statement Claiming Small Entity Status.

10. ☐ Microfiche Computer Program (Appendix).

**NEW UNITED STATES UTILITY PATENT APPLICATION**  
**under 37 C.F.R. 1.53(b)**

Page 2

**Atty. Docket No. 03797.84196**

11. ☐ Nucleotide and/or Amino Acid Sequence Submission.  
☐ Computer Readable Copy.  
☐ Paper Copy (identical to computer copy).  
☐ Statement verifying identity of above copies.

**NEW UNITED STATES UTILITY PATENT APPLICATION**  
**under 37 C.F.R. 1.53(b)**

Page 3

Atty. Docket No. 03797.84196

12. Calculation of Fees:

FEES FOR	EXCESS CLAIMS	FEE	AMOUNT DUE
Basic Filing Fee (37 C.F.R. § 1.16(a))			\$690.00
Total Claims in Excess of 20 (37 C.F.R. § 1.16(c))	16	18.00	\$288.00
Independent Claims in Excess of 3 (37 C.F.R. § 1.16(b))	4	78.00	\$312.00
Multiple Dependent Claims (37 C.F.R. § 1.16(d))	0	260.00	\$0.00
Subtotal - Filing Fee Due			\$1,290.00
	MULTIPLY BY		
Reduction by 50%, if Small Entity (37 C.F.R. §§ 1.9, 1.27, 1.28)	0		\$0.00
<b>TOTAL FILING FEE DUE</b>			<b>\$1,290.00</b>
Assignment Recordation Fee (if applicable) (37 C.F.R. § 1.21(h))	1	40.00	\$40.00
<b>GRAND TOTAL DUE</b>			<b>\$1,330.00</b>

13. PAYMENT is:

- ☒ included in the amount of the GRAND TOTAL by our enclosed check. A general authorization under 37 C.F.R. § 1.25(b), second sentence, is hereby given to credit or debit our Deposit Account No. 19-0733 for the instant filing and for any other fees during the pendency of this application under 37 C.F.R. §§ 1.16, 1.17 and 1.18.
- ☐ not included, but deferred under 37 C.F.R. § 1.53(f).

14. All correspondence for the attached application should be directed to:

Pamela I. Banner  
Banner & Witcoff, Ltd.  
1001 G. Street N.W., 11th Floor  
Washington, D.C. 20001-4597  
Telephone: (202) 508-9100  
Facsimile: (202) 508-9299

15. Other: \_\_\_\_\_

Date: April 27, 2000

By:



Christopher P. Moreno  
Reg. No. 38,566

CPM/dlt

## Creation and Use of Virtual Device Drivers on a Serial Bus

### Technical Field

The present invention relates generally to the use of serial buses as a means of communication between electronic devices and, in particular, to virtual device driver implementation on a serial bus, such as a serial bus operating in conformance with the IEEE 1394 Serial Bus standard.

### Background of the Invention

Computer systems are typically comprised of a variety of different components or devices that operate together to form the resultant system. Some of the devices are supplied with the computer system initially, such as the central processing unit, and some devices can be installed into the computer system after the initial configuration of the system. The devices of the computer system are generally coupled together via interconnects which may be of several types, such as a serial bus.

Serial buses are well known in the art. A recently developed serial bus standard is the IEEE 1394 serial bus standard, disclosed in the ISO/IEC 13213 (ANSI/IEEE 1212) CSR Architecture Specification and the IEEE 1394-1995 Serial Bus Specification, the teachings of which are herein incorporated by this reference. A typical serial bus having an IEEE 1394 standard architecture is comprised of a multiplicity of nodes that are interconnected via point-to-point links, such as cables, that each connect a single node of the serial bus to another node of the serial bus. Each node is an addressable entity that can be reset and identified. Nodes are associated with respective components of the computer system and serve as interfaces between the components and communication links. Each node has a configuration ROM (CROM), the registers of which can be accessed by software residing within the computer system. The IEEE 1394 standard sets forth a general CROM format which comprises several fields. One field in particular is the unit directory. The unit directory contains information representing the functionality of units within the node, particularly the unit's software version number and its location within the node. Generally, the information in the configuration ROM is treated as static. However, U.S. Patent Application serial no. 09/441,264 in the name of G. Chrysanthakopoulos and entitled "Modification and Use of Configuration Memory Used

During Operation of a Serial Bus” provides a technique for dynamically changing the configuration ROM, the teachings of which are herein incorporated by this reference. This patent application describes a technique of creating multiple unit directories for multiple device representation.

5 Device drivers are well known in the art. When a user installs a new device on to a computer system, a device driver is loaded for communication with the device. A device driver is software within an operating system that controls a device. A virtual device driver is a special type of device driver that has full access to the operating system kernel and can communicate directly to a physical port but was loaded without a  
10 hardware device being detected or enumerated by the system. A virtual device driver manipulates kernel mode code using existing hardware resources to emulate a device that is not normally present on the computer. In connection with a 1394 serial bus, a virtual driver is given more access than a traditional device driver because it is not restricted to talking to just one particular device.

15 Virtual device drivers are designed to handle hardware device contention between multiple processes and to translate or buffer data transfers from a virtual machine to hardware devices. A virtual machine is a self-contained operating environment that behaves as if it were a separate computer. When two or more processes attempt to access the same device, some method of contention management must be used. A virtual device  
20 driver allows each process to act as though it has exclusive access to the device. For example, a virtual printer driver would provide the printing process with a virtual printer port, and characters written to the port would be written to a print spooler. The virtual device driver would then send the job to the printer when it becomes available. Another method would be to assign the physical device to only one process at a time, so that when  
25 a process attempts to access the device while it is in use, the virtual device driver does not pass the request to the actual hardware, and the process operates as though the hardware did not exist.

Recently, virtual device drivers have been expanded to include interprocess communication. Virtual device drivers can provide the necessary mechanisms to allow  
30 a virtual machine to see a device that may not actually exist in hardware. Virtual device drivers can also implement client-server hardware management by providing an interface

to a virtual machine. Virtual device drivers also virtualize input/output to the device and translate this information into commands to be sent across a network to a hardware server.

Currently, when a device is plugged into a personal computer (PC) on a 1394 bus, the 1394 bus driver interface creates a device object. Based on the device object, the so-called plug and play (PnP) subsystem loads high level device drivers that facilitate communication between the user and the device. At this time, the PC does not emulate any device, rather the PC exposes a generic CROM on the 1394 bus. Other nodes on the bus use the CROM to detect that the PC is in fact a PC. Enumeration occurs when a node on a serial bus accesses the configuration memory of another node to see what functionality the node has. The node accessing the CROM would then load a device object and device driver according to what functionality was exposed in the configuration memory. A technique that allows device emulation on a hardware platform that runs a general purpose operating system is not currently known in the art. However, such a technique would offer significant advantages over the prior art.

A further problem with current technology is the inability of devices to communicate natively, i.e., without translations, over a serial bus. For example, streaming video between two or more PCs will typically require translations at the transmitting and receiving ends of a serial bus. A first PC may have an audio video interleave (AVI) file that it wishes to send to a second PC. AVI is currently a video standard in a "WINDOWS" brand operating system. In order to send the file, the first PC would have to convert the AVI file to network packets and then stream it over the Internet protocol (IP) network. A technique that allows devices to transmit such files over a serial bus without converting the files is not currently known in the art. However, such a technique would offer significant advantages over the prior art.

### **Summary of the Invention**

The present invention provides a way for a node, such as a personal computer (PC), on a serial bus to emulate any desired device using virtual device drivers. A PC connected to a 1394 bus exposes its CROM on the bus. The CROM presents an image to other nodes on the 1394 bus and describes the functional units supported by the node. A

software driver or a virtual device driver changes the CROM by adding a unit directory detailing a connected device that the node will emulate. The CROM is changed dynamically by adding unit directories to the CROM detailing peripherals connected to the PC. By changing the CROM, the PC can be enumerated as the connected device by other PCs on the bus. In this manner, the PC can instantly emulate or morph itself into any desired device. The PC can also emulate multiple devices at the same time. Therefore, the present invention is particularly useful where multiple emulation drivers need to coexist. The PC can also create virtual device objects for devices that don't yet exist on the bus. A user can trigger the creation of virtual device objects with device properties for devices that are not currently connected to the PC or are present on the 1394 serial bus. The PC may then emulate any device automatically with or without a physical device present.

### **Brief Description of the Figures**

FIG. 1 is a block diagram of an exemplary operating environment.

FIG. 2 is a block diagram of a system through which the present invention may be implemented.

FIG. 3 is a flow chart illustrating a method of emulating a device in accordance with the present invention.

FIG. 4 is a flow chart illustrating a method of creating a virtual device in accordance with the present invention.

FIG. 5 is a flow chart illustrating a method of removing a virtual device in accordance with the present invention.

FIG. 6 is a flow chart illustrating a method of implementing an emulation driver in accordance with the present invention.

### **Detailed Description of the Invention**

The present invention may be more fully described with reference to FIGS. 1-6.

FIG. 1 is a schematic diagram of a conventional general-purpose digital computing environment that can be used to implement various aspects of the invention. Computer 100 includes a processing unit 110, a system memory 120 and a system bus

130 that couples various system components including the system memory to the processing unit 110. System bus 130 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory 120 includes a read only memory (ROM) 140 and a random access memory (RAM) 150.

A basic input/output system (BIOS) 160 containing the basic routines that help to transfer information between elements within the computer 100, such as during start-up, is stored in ROM 140. Computer 100 also includes a hard disk drive 170 for reading from and writing to a hard disk (not shown), a magnetic disk drive 180 for reading from or writing to a removable magnetic disk 190, and an optical disk drive 191 for reading from or writing to a removable optical disk 192, such as a CD ROM or other optical media. Hard disk drive 170, magnetic disk drive 180, and optical disk drive 191 are respectively connected to the system bus 130 by a hard disk drive interface 192, a magnetic disk drive interface 193, and an optical disk drive interface 194. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for personal computer 100. It will be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

A number of program modules can be stored on the hard disk, magnetic disk 190, optical disk 192, ROM 140 or RAM 150, including an operating system 195, one or more application programs 196, other program modules 197, and program data 198. A user can enter commands and information into computer 100 through input or selection devices, such as a keyboard 101 and a pointing device 102. The pointing device 102 may comprise a mouse, touch pad, touch screen, voice control and activation or other similar devices. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 110 through a serial port interface 106 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, a game port or a



universal serial bus (USB). A monitor 107 or other type of display device is also connected to system bus 130 via an interface, such as a video adapter 108. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

5 An additional serial port in the form of an IEEE 1394 interface 140 may also be provided. The IEEE 1394 interface 140 couples an IEEE 1394-compliant serial bus 145 to the system bus 130 or similar communication bus. The IEEE 1394-compliant serial bus 145, as known in the art, allows multiple devices 150 to communicate with the computer 100 and each other using high-speed serial channels.

10 Computer 100 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 109. Remote computer 109 typically includes at least some of the elements described above relative to computer 100, although only a memory storage device 111 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 112 and a wide area network (WAN) 113. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

15 When used in a LAN networking environment, computer 100 is connected to local network 112 through a network interface or adapter 114. When used in a WAN networking environment, personal computer 100 and remote computer 109 may both include a modem 115 or other means for establishing a communications over wide area network 113, such as the Internet. Modem 115, which may be internal or external, is connected to system bus 130 via serial port interface 106. In a networked environment, program modules depicted relative to personal computer 100, or portions thereof, may be stored in the remote memory storage device.

25 It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers can be used. The existence of any of various well-known protocols, such as TCP/IP, "ETHERNET", FTP, HTTP and the like, is presumed, and the system can be operated in a client-server configuration to permit a user to retrieve web pages from a web-based server. For example, in an embodiment of the present invention, the remote computer

30

109 is a server having stored thereon one or more documents that may be accessed by the computer 100.

Procedures of the present invention described below can operate within the environment of the computer shown in FIG. 1. Although the present invention is generally applicable to a computer operating in accordance with the IEEE 1394 standard, the present invention is applicable to any computer system that implements the Control and Status Registers (CSR) configuration ROM architecture described in the IEEE 1212R CSR Architecture Specification.

In FIG. 2, there is a system that may be used to implement the present invention. A personal computer (PC1) 200 may be connected to a 1394 serial bus 202. PC1 200 comprises a 1394-compliant bus driver 204, which manages communications between the physical bus 202 and higher level protocol layers. PC1 200 also has a configuration memory 206 which exposes PC1's functionality on the serial bus 202. A user of PC1 200 has the option of creating a virtual device object (VDO) 212 to represent a device capable of being plugged into a PC such as a printer, scanner, DVD drive, camcorder, or the like. The VDO 212 then loads an emulation driver 214 appropriate for the device being emulated. The VDO 212 and emulation driver 214 remain present even if PC1 is rebooted. The emulation driver 214 is in communication with and can alter the configuration memory (CROM) 206 to add a unit directory 216. The unit directory 216 represents the functionality of the emulated device. The CROM 206 exposes the functionality of the device on the serial bus 202. The user may want to emulate more than one device. In this case, the user would repeat the process by creating a second VDO (not shown) with the target functionality of the newly emulated device. The second VDO would then load a second emulation driver (not shown). Several VDOs 212 and emulation drivers 214 can be created and can exist at the same time. The emulation drivers 214 continue to add unit directories to the CROM - one unit directory for each device the user wishes to emulate. PC1 200 can then emulate as many devices as it has unit directories 216.

One benefit of the present invention is that it instantly allows a PC to emulate multiple devices at the same time. Another benefit of the present invention is that it does not require that a device or another PC be plugged in to create a VDO. A user mode

application sends a request that tells the 1394 bus driver to create a VDO with certain properties. The VDO can be created to have just in case the device is ever plugged in. The VDO loads an emulation driver that supports the target functionality of the device or implements the complete set of features, of a 1394 device. If another PC is plugged into the PC, the VDO is already present and is immediately capable of representing the complete functionality of the emulated device to another PC, or other node on the serial bus. Formerly, the PC would not be able to represent to other nodes on the serial bus functionality other than that of a physical device attached to the node.

The dotted lines in FIG. 2 represent optional elements. For purposes of a second illustration, a device 208 may be connected to PC1 200. The device 208 could be any device capable of being plugged into a PC such as a printer, a scanner, a DVD drive, or the like. For this example, the device 208 is assumed to be a USB printer. PC1 200 would have a device driver (USB printer driver) 210 that enables communication with the device 208. The user can create a VDO 212 that represents a 1394 printer even though a 1394 compliant printer is not attached to PC1 200. The user may create a VDO by modifying installation files. When a 1394 controller is detected, a VDO entry is automatically created in the registry. The VDO 212 then loads an emulation driver 214 for communication with the device 208. The emulation driver 214 actually communicates with the (USB printer) device driver 210. The VDO 212 and emulation driver 214 remain present even if the device 208 is unplugged. The emulation driver 214 is also in communication with the configuration memory 206 and can alter the configuration memory 206 by adding a unit directory 216. The unit directory 216, in accordance with the 1394 standard, describes the functionality of a device, in this case a 1394 printer.

Another node may be present on the serial bus 202, for example, a second PC (PC2) 220. When enumerating other nodes on the serial bus 202, PC2 220 accesses the configuration memory 206 of PC 200 and reads the unit directory 216 detailing the emulated device. In response to the functionality exposed in the unit directory, PC2 220 creates a physical device object (PDO) 222 for the device, a "1394 printer." PC2 then loads the appropriate device driver 224 for communication with the "1394 printer."

In addition to being able to emulate multiple devices at one time and not requiring that a device be plugged in to emulate the device, another benefit of the present invention is that it allows “native” communication across the serial bus. In the previous example, PC2 can communicate using “native” language because it believes it is communicating with a 1394 printer instead of a USB printer. No translations are necessary because PC1’s emulation driver 214 communicates directly with the USB device driver 210.

In FIG. 3, a method of emulating a device is shown. At step 300, a virtual device object is created by the 1394 bus driver. This step will be discussed in further detail in connection with FIG. 4. Then, the appropriate emulation driver relating to the device is loaded at step 302. The emulation driver has the ability to communicate with and alter the configuration memory to add device specific details to the configuration memory. The configuration memory exposes functionality of the device being emulated on the serial bus at step 304. This process can be repeated several times for each device the PC is to emulate.

The dotted portion of FIG. 3 represents optional steps. After the device functionality is exposed at step 304 on the serial bus, a bus reset can be forced. This bus reset causes all devices or nodes attached to the serial bus to enumerate each other at step 306. Any other node may now see the node with altered configuration memory as the device it has chosen to emulate. The other node then creates a physical device object for the device at step 308 and may load the appropriate device driver at step 310.

In FIG. 4, a method of creating a virtual device is shown. At step 400, a request in the form of a data structure is sent to the application program interface (API). The request can be sent by an upper level driver that is already loaded for a 1394 device but now it also wants to emulate a device. The request could also be sent by an application upon user request. A user might want to make his/her PC look like a DVD drive so other nodes on the 1394 bus can use it to store and retrieve data from the user’s internal 1394 DVD. Using the IOCTL\_IEEE1394\_API\_REQUEST, software can pass the following data structure to the 1394 bus driver:

```
typedef struct _IEEE1394_API_REQUEST {  
    ULONG    RequestNumber;  
    ULONG    Flags;
```

```

union      {
    ...
} u;
} IEEE1394_API_REQUEST, *PIEEE1394_API_REQUEST;

```

The data structure is comprised of at least two fields. The first field within the data structure is configured to add a virtual device by configuring IEEE1394\_API\_REQUEST.RequestNumber = IEEE1394\_API\_ADD\_VIRTUAL\_DEVICE.

IEEE1394\_API\_ADD\_VIRTUAL\_DEVICE is further defined by the following data structure:

```

typedef struct _IEEE1394_VDEV_PNP_REQUEST {
    ULONG      FulFlags;
    ULONG      Reserved;
    ULARGE_INTEGER InstanceId;
    UCHAR      DeviceId;
} IEEE1394_VDEV_PNP_REQUEST, *PIEEE1394_VDEV_PNP_REQUEST;

```

Once the API\_REQUEST is configured to add a virtual device, then the device data structure is filled in. FulFlags is a flag that can be configured if the text string is in Unicode by setting IEEE1394\_VDEV\_PNP\_REQUEST.FulFlags = IEEE1394\_REQUEST\_FLAG\_UNICODE. InstanceId is a 64-bit number that can be used to identify this instance of the virtual device. DeviceId is a null terminated string to be used for generating the PnP ids required to enumerate the emulation driver.

The second field is a flag. The second field within the data structure is configured at step 402 to allow the virtual device to remain present despite a subsequent hardware or software reboot by configuring IEEE1394\_API\_REQUEST.Flags = IEEE1394\_API\_FLAG\_PERSISTANT. This will guarantee that this VDO will be reported after a reboot. Then at step 404, the API request is sent to the 1394 bus driver.

In FIG. 5, a method of removing a virtual device is shown. At step 500, an API request data structure is set up and the first field is configured to remove a virtual device (rather than add a device as in FIG. 4). Using the data structure described in reference to Fig. 4, the IEEE1394\_API\_REQUEST.RequestNumber =

IEEE1394\_API\_REMOVE\_VIRTUAL\_DEVICE. Then at step 502, the API request data structure is sent to the 1394 bus driver. Because the IEEE1394\_API\_REQUEST.Flags is configured to allow the virtual device to remain present over boots when the virtual device object is added, this request is sent to remove the virtual device. The request can be a request to remove the virtual device or it can be a request to remove an entry from the registry. Existing PnP methods can also be used to remove the VDO. An IRP\_MN\_REMOVE\_DEVICE can be sent to the driver stack enumerated on the VDO.

In FIG. 6, a method for implementing an emulation driver is shown. At step 600, the configuration memory is modified. One embodiment of the invention allows the configuration memory to be modified wherein the VDO submits a request to modify by using the SET\_LOCAL\_HOST\_PROPERTIES\_MODIFY\_CROM request. A unit directory, in conformance with the IEEE 1394 standard, is then added to the configuration memory with the details of the device. All information of the emulated device functionality is then added or altered to expose that functionality on the serial bus.

Then, at step 602 a bus reset is issued. This step is performed to cause all nodes on the serial bus to re-enumerate each other. Any other node on the bus can then access the configuration memory and see the details of the device. The other node's operating system believes the emulated device is present. In other words, the other node can then "see" the node as the emulated device. The benefit of such a process is that the node is actually being seen as the device, rather than having a device connected to it, as was done in the past. This is a benefit because it would allow any other node on the bus to communicate "natively" with the device rather than using the node as a server/translator for the device. Then, at step 604, node address space is allocated in order to intercept requests to an emulated device register by using the REQUEST\_ALLOCATE\_ADDRESS. To allow any external device to access those addresses, the ACCESS\_FLAG\_BROADCAST must be set when allocating the addresses.

Generally, VDOs and the respective drivers have the same access to the 1394 bus driver as would a physical device object and its respective driver. However, there are differences in behavior with a VDO because there is no physical target device. Normally,

the 1394 bus driver fills in the target node identifier and the appropriate packet size and transfer rate using information from the enumeration procedure with a particular device. However, in the present invention the VDO must provide all packet information because there is no target device node. For example, a

5 REQUEST\_ASYNC\_READ/WRITE/LOCK will be intercepted and the VDO will fill in the address information for the request. The bus driver makes sure not to overwrite any fields. REQUEST\_ALLOCATE\_RANGE also exhibits different behavior if addressed to a VDO. All address allocations from an emulation driver will implicitly have the ACCESS\_FLAG\_BROADCAST enabled if post notification on the address range is

10 required. This is done to allow any external node to access the address range used by the emulation driver to simulate the device. Similarly, there are requests that will not be supported because there is no device. For example, the requests REQUEST\_GET\_ADDR\_FROM\_DEVICE\_OBJECT and REQUEST\_SET

15 DEVICE\_XMIT\_PROPERTIES are not supported for virtual devices because there is no corresponding hardware node. For all other requests, the behavior is identical between virtual and physical devices.

Although the invention has been described in relation to preferred embodiments, many variations, equivalents, modifications and other uses will become apparent to those skilled in the art. The scope of the present invention should not be limited to the specific

20 disclosure but determined only by the appended claims.

## Claims

What is claimed is:

1. A method of emulating a device by a node on a serial bus comprising the steps of:  
creating a virtual device object for the device;  
responsive to the step of creating the virtual device object, loading an emulation  
driver for the device; and  
5 dynamically exposing, on the serial bus, an emulated device functionality.
2. The method of claim 1 further comprising a step of: enumerating, by the node, at least  
one other node on the serial bus.
3. The method of claim 2 further comprising the steps of:  
creating, by the at least one other node, a physical device object for the device; and  
loading a device driver for the device.
4. The method of claim 1 wherein the step of creating the virtual device object is done  
by a bus driver.
5. The method of claim 4 wherein the bus driver is an IEEE 1394 compliant bus driver.
6. The method of claim 1 wherein the device is capable of being plugged natively into  
the serial bus.
7. The method of claim 1 wherein the serial bus is an IEEE 1394 compliant serial bus.
8. The method of claim 1 wherein the virtual device object can exist independent of bus  
events.
9. The method of claim 8 wherein the bus events include at least one of: addition of the  
device and removal of the device.
10. The method of claim 1 wherein the node is a personal computer running a general  
purpose operating system.
11. The method of claim 1 wherein the step of exposing the emulated device functionality  
is done by configuration memory.
12. A method for creating a virtual device object comprising:  
creating a data structure which comprises a request for creating a virtual device  
object; and  
sending the data structure to a bus driver which, responsive to the request, creates the  
5 virtual device object.



13. The method of claim 12 wherein the data structure comprises:

a request field; and

a flag field.

14. The method of claim 12 wherein the bus driver is an IEEE 1394 compliant bus driver.

~~15.~~ A method for removing a virtual device comprising:

creating a data structure which comprises a request for removing a virtual device object; and

5 sending the data structure to a bus driver which, responsive to the request, removes the virtual device object.

16. The method of claim 15 wherein the bus driver is an IEEE 1394 compliant bus driver.

~~17.~~ A method for implementing an emulation driver comprising:

modifying a configuration memory;

issuing a bus reset; and

allocating node address space to intercept requests to an emulated device register.

18. The method of claim 17 wherein modifying a configuration memory further comprises:

submitting a request to modify by a virtual device object;

adding a unit directory to the configuration memory; and

5 altering information necessary to expose an emulated device functionality.

~~19.~~ A system for emulating a device, comprising in combination:

a serial bus; and

a node connected to the serial bus which will emulate at least one device.

20. The system of claim 19 wherein the serial bus is an IEEE-1394 compliant serial bus.

21. The system of claim 19 wherein the node further comprises:

a configuration memory compliant with IEEE-1212 standard in which device functionality will be stored; and

a layered protocol stack.

22. The system of claim 21 wherein the layered protocol stack further comprises:

a bus driver for controlling bus communications;

at least one device object in communication with the bus driver for representing the at least one device; and

at least one device driver in communication with the at least one device object for interfacing with the at least one device.

23. The system of claim 22 wherein the bus driver is an IEEE-1394 compliant bus driver.

24. The system of claim 22 wherein the device object is a virtual device object.

25. The system of claim 22 wherein the device driver is a virtual device driver.

26. The system of claim 19 which further comprises a physical device coupled to the node.

27. A device for emulating at least one other device comprising:  
a configuration memory compliant with IEEE-1212 standard; and  
a layered protocol stack in communication with the configuration memory.

28. The device of claim 27 wherein the layered protocol stack furthers comprises:

a bus driver;

at least one device object in communication with the bus driver for representing the at least one other device; and

5 at least one device driver in communication with the at least one device object for interfacing with the at least one other device.

29. The device of claim 27 wherein the bus driver is an IEEE-1394 compliant bus driver.

30. The device of claim 27 wherein the device object is a virtual device object.

31. The device of claim 27 wherein the device driver is a virtual device driver.

32. The device of claim 27 wherein the configuration memory has at least one unit directory.

33. A computer-readable medium comprising instructions that, when executed by a computer on which a device will be emulated, perform the steps of:

creating a virtual device object;

loading an emulation driver; and

5 dynamically exposing, on a serial bus, a device functionality.

34. The computer-readable medium of claim 33 wherein the computer instructions when executed further perform the step of causing enumeration, by the computer, of at least one other node on the serial bus.

35. The computer-readable medium of claim 34 wherein the enumeration is caused by a bus reset.

36. The computer-readable medium of claim 33 wherein the serial bus is an IEEE-1394 compliant serial bus.

## Abstract of the Invention

A node on a serial bus, preferably a device such as a personal computer (PC), can emulate other devices using virtual device drivers. A PC connected to a 1394 bus exposes its CROM on the bus which presents an image to other nodes on the 1394 bus and describes the functional units supported by the node. The CROM can be changed dynamically by adding unit directories to the CROM detailing peripherals connected to the PC. The PC can then be enumerated as the connected device by other PCs on the bus. The PC can emulate or morph itself into any desired device or even multiple devices at the same time. The invention also allows a PC to create devices that don't yet exist on the bus. The invention allows a user to create virtual device objects with device properties to have just in case a user plugs the particular device in to the PC.

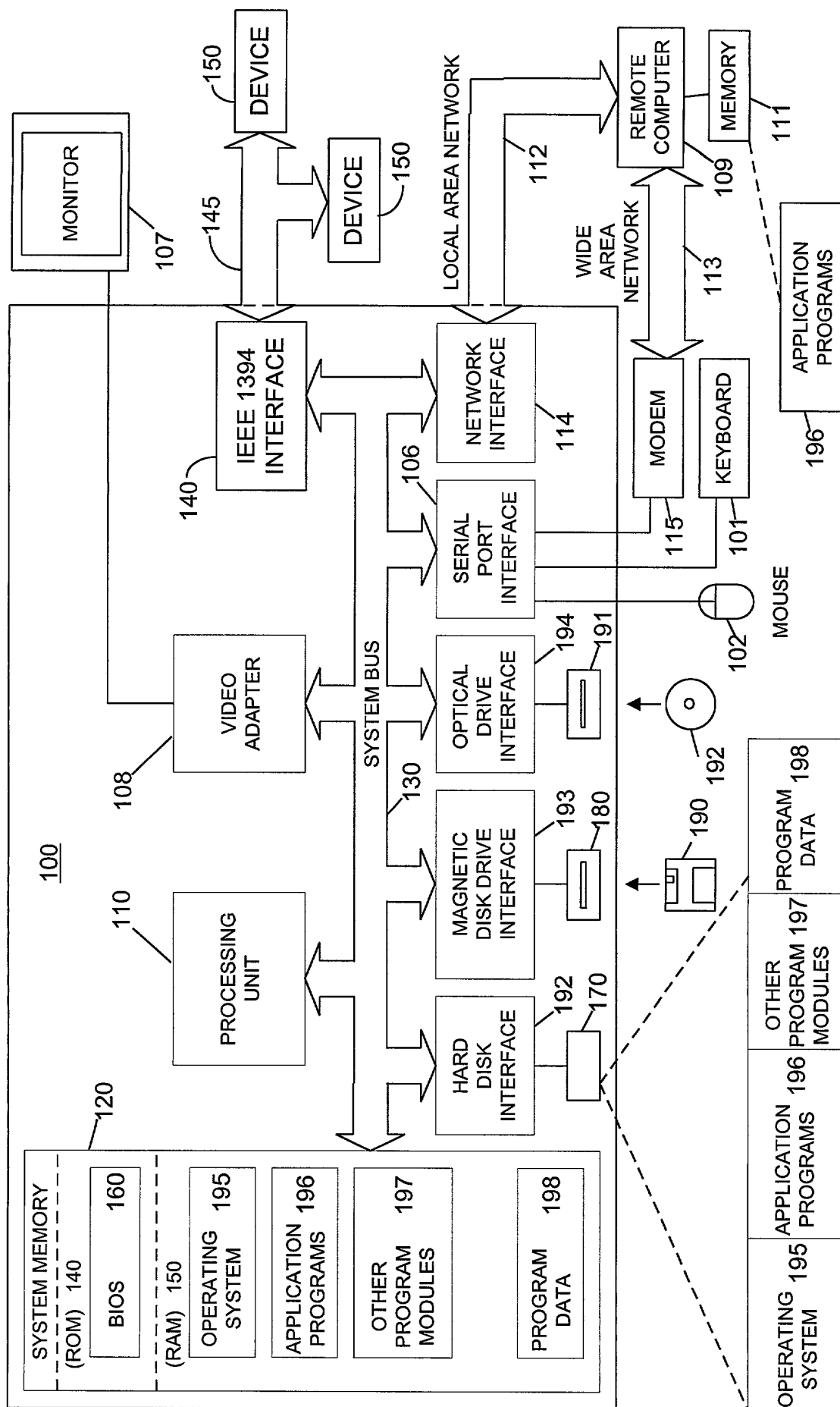


FIG. 1

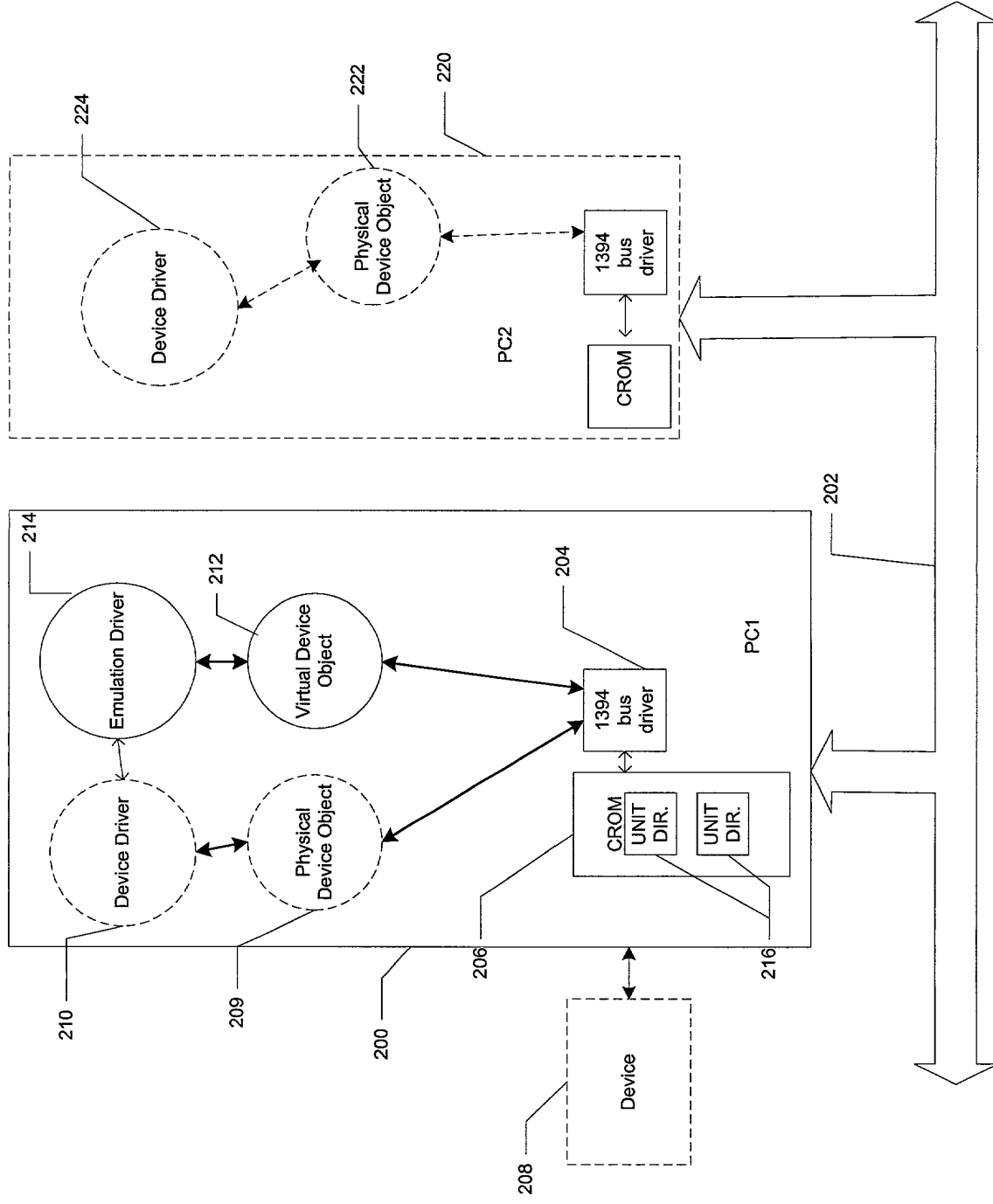


Figure 2

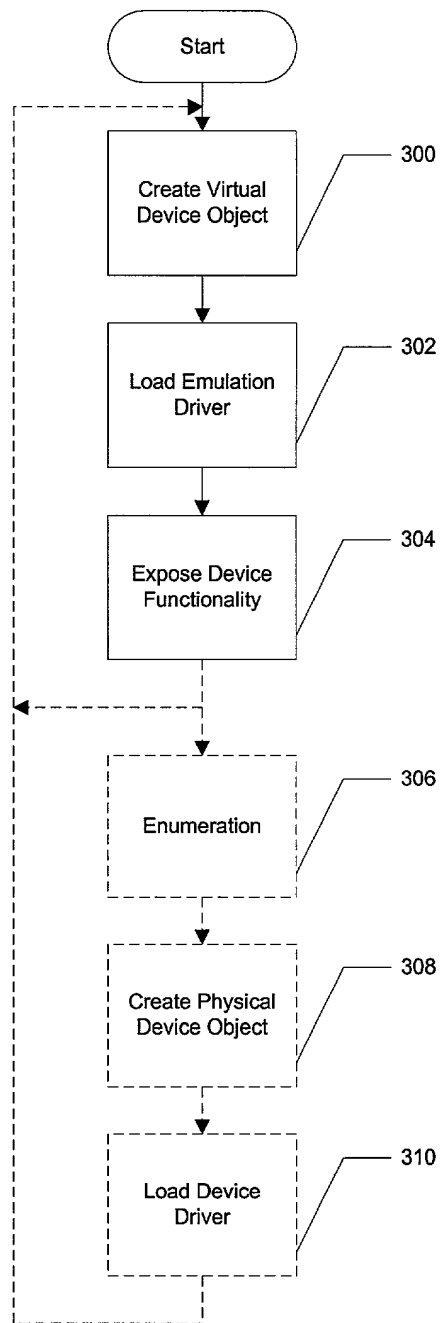


Figure 3

Figure 4

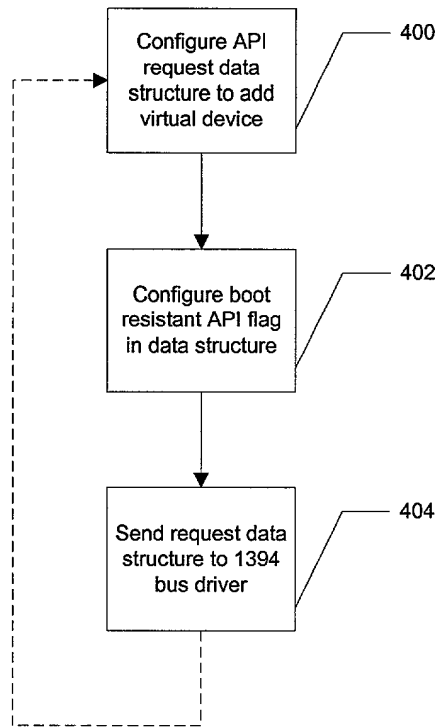
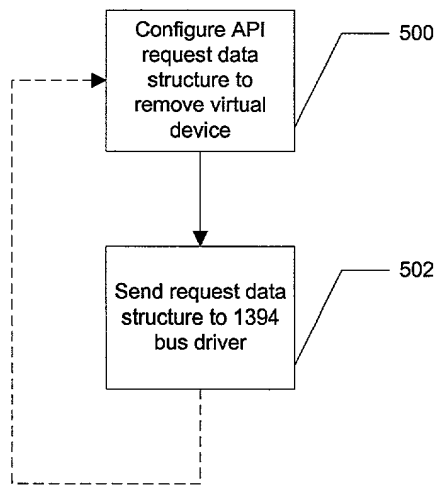


Figure 5





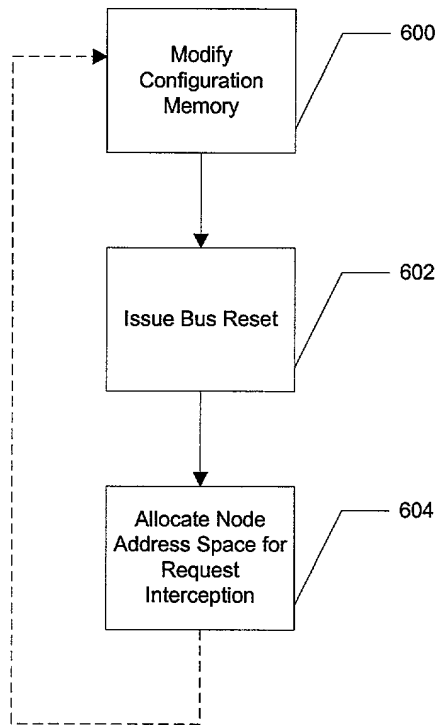


Figure 6

# SOLE DECLARATION FOR PATENT APPLICATION

As the below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled Creation and Use of Virtual Device Drivers on a Serial Bus, the specification of which

- ☒ is attached hereto.  
☐ was filed on \_\_\_\_\_ as Application Serial Number \_\_\_\_\_ and was amended on (if applicable).  
☐ was filed under the Patent Cooperation Treaty (PCT) and accorded International Application No. \_\_\_\_\_, filed \_\_\_\_\_, and amended on \_\_\_\_\_ (if any).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I hereby acknowledge the duty to disclose information which is material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56(a).

## Prior Foreign Application(s)

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application(s) for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Country	Application No.	Date of Filing (day month year)	Date of Issue (day month year)	Priority Claimed Under 35 U.S.C. §119

## Prior United States Provisional Application(s)

I hereby claim priority benefits under Title 35, United States Code, §119(e)(1) of any U.S. provisional application listed below:

U.S. Provisional Application No.	Date of Filing (day month year)	Priority Claimed Under 35 U.S.C. §119(e)(1)

## Prior United States Application(s)

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, §1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Application Serial No.	Date of Filing (Day, Month, Year)	Status — Patented, Pending, Abandoned

### Power of Attorney

And I hereby appoint, both jointly and severally, as my attorneys with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith the following attorneys and agents, their registration numbers being listed after their names:

ALTHERR, Robert F.	31,810	HOSCHEIT, Dale H.	19,090	PATHAK, Ajay S.	38,266
BANNER, Donald W.	17,037	IWANICKI, John P.	34,628	PAYNE, Stephen S.	35,316
BANNER, Mark T.	29,888	JACKSON, Thomas H.	29,808	PETERSON, Thomas L.	30,969
BANNER, Pamela I.	33,644	KAGAN, Sarah A.	32,141	POTENZA, Joseph M.	28,175
BECKETT, William W.	18,262	KATZ, Robert S.	36,402	PRATT, Thomas K.	37,210
BODNER, Jordan	42,338	KLEIN, William J.	43,719	RENK, Christopher J.	33,761
BUROW, Scott A.	42,373	KRAUSE, Joseph P.	32,578	RESIS, Robert H.	32,168
CALLAHAN, James V.	20,095	LINEK, Ernest V.	29,822	RIVARD, Paul M.	43,446
CHANG, Steve S.	42,402	MAGOON, Sumeet	43,769	SCHAD, Steve P.	32,550
COHAN, Gregory J.	40,959	MALONE, Dale A.	32,155	SHANAHAN, Michael H.	24,438
COOPERMAN, Marc S.	34,143	MANNAVA, Ashok K.	45,301	SHIPLEY, Charles W.	28,042
CURTIN, Joseph P.	34,571	McDERMOTT, Peter D.	29,411	SKERPON, Joseph M.	29,864
DAWSON, John R.	39,504	McKEE, Christopher L.	32,384	SPAR, Elizabeth	45,123
DEMOOR, Laura J.	39,654	McKIE, Edward F.	17,335	STOCKLEY, D. J.	34,257
EVANS, Thomas L.	35,805	MEDLOCK, Nina L.	29,673	VAN ES, J. Pieter	37,746
FEDOROCHKO, Gary D.	35,509	MEECE, Timothy C.	38,553	WILLIAMS, Kathleen M.	34,380
FISHER, Daniel E.	34,162	MEEKER, Frederic M.	35,282	WITCOFF, Sheldon W.	17,399
FISHER, William J.	32,133	MILLER, Charles L.	43,805	WOLFF, Kevin A.	42,233
GLEMBOCKI, Christopher R.	38,800	MITRIUS, Janice V.	43,808	WOLFFE, Franklin D.	19,724
HANLON, Brian E.	40,449	MORENO, Christopher P.	38,566	WOLFFE, Susan A.	33,568
HEMMENDINGER, Lisa M.	42,653	NELSON, Jon O.	24,566	WRIGHT, Bradley C.	38,061
HILLMAN, Lisa	43,673	NIEGOWSKI, James A.	28,331	SAKO, Katie E.	32,628
HONG, Patricia E.	34,373	PATEL, Binal J.	42,065	CROUSE, Daniel D.	32,022

All correspondence and telephone communications should be addressed to:

Pamela I. Banner

Banner & Witcoff, Ltd.

1001 G Street, N.W., 11th Floor

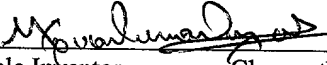
Washington, D.C. 20001-4597

Customer Number: 22907

Tel: (202) 508-9100

Fax: (202) 508-9299

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Signature  Date 3/16/2000  
Full Name of Sole Inventor Chrysanthakopoulos Georgios  
Family Name First Given Name Second Given Name  
Residence Kirkland, Washington Citizenship Greece  
Post Office Address 715 7<sup>th</sup> Street South, Kirkland, Washington 98033